




# ¿Qué es la seguridad de la memoria y por qué es importante?

La seguridad de la memoria es una propiedad de algunos lenguajes de programación que evita que los programadores introduzcan ciertos tipos de errores relacionados con el uso de la memoria. Dado que los errores de seguridad de la memoria suelen ser problemas de seguridad, los idiomas seguros para la memoria son más seguros que los idiomas que no lo son.

Los lenguajes seguros para la memoria incluyen Rust, Go, C#, Java, Swift, Python y JavaScript. Los lenguajes que no son seguros para la memoria incluyen C, C++ y ensamblador.

## Tipos de errores de seguridad de la memoria

Para comenzar a comprender los errores de seguridad de la memoria, consideraremos el ejemplo de una aplicación que mantiene listas de tareas para muchos usuarios. Veremos algunos de los tipos más comunes de errores de seguridad de la memoria que pueden ocurrir en programas que no son seguros para la memoria.

### Lecturas y escrituras fuera de los límites

Si tenemos una lista de tareas con diez elementos y pedimos el undécimo elemento, ¿qué debería suceder? Claramente deberíamos recibir un error de algún tipo. También deberíamos obtener un error si solicitamos el primer elemento negativo.

En estas circunstancias, un lenguaje que no es seguro para la memoria puede permitir que un programador lea cualquier contenido de memoria que exista antes o después de los contenidos válidos de la lista. Esto se denomina lectura fuera de los límites. El recuerdo anterior al primer elemento de una lista podría ser el último elemento de la lista de otra persona. El recuerdo después del último elemento de una lista podría ser el primer elemento de la lista de otra persona. ¡Acceder a esta memoria sería una grave vulnerabilidad de seguridad! Los programadores pueden evitar lecturas fuera de los límites comprobando diligentemente el índice del elemento que solicitan con la longitud de la lista, pero los programadores cometen errores. Es mejor usar un lenguaje seguro para la memoria que lo proteja a usted y a sus usuarios de la clase de errores de manera predeterminada.

En un lenguaje seguro para la memoria, obtendremos un error en tiempo de compilación o un bloqueo en tiempo de ejecución. Bloquear el programa puede parecer grave, ¡pero es mejor que dejar que los usuarios roben los datos de los demás!

Una vulnerabilidad estrechamente relacionada es una escritura fuera de los límites. En este caso, imagine que tratamos de cambiar el undécimo o el primer elemento negativo en nuestra lista de tareas pendientes. ¡Ahora estamos cambiando la lista de tareas de otra persona!

### Usar después gratis

Imagine que eliminamos una lista de tareas pendientes y luego solicitamos el primer elemento de esa lista. Claramente, deberíamos recibir un error, ya que no deberíamos poder obtener elementos de una lista eliminada. Los lenguajes que no son seguros para la memoria permiten que los programas obtengan la memoria con la que han dicho que han terminado, y que ahora puede usarse para otra cosa. ¡La ubicación en la memoria ahora puede contener la lista de tareas de otra persona! Esto se denomina vulnerabilidad de uso después de la liberación.

## ¿Qué tan comunes son las vulnerabilidades de seguridad de la memoria?

Extremadamente. Un estudio reciente encontró que el 60-70 % de las vulnerabilidades en iOS y macOS son vulnerabilidades de seguridad de la memoria. Microsoft estima que el 70% de todas las vulnerabilidades en sus productos durante la última década han sido problemas de seguridad de la memoria. Google estimó que el 90% de las vulnerabilidades de Android son problemas de seguridad de la memoria. Un análisis de los días 0 que se descubrieron explotados en la naturaleza encontró que más del 80 % de las vulnerabilidades explotadas eran problemas de seguridad de la memoria [1] .

El gusano Slammer de 2003 fue un desbordamiento de búfer (escritura fuera de los límites). También WannaCry (escritura fuera de los límites). El exploit Trident contra iPhones usó tres vulnerabilidades de seguridad de memoria diferentes (dos uso después de liberar y una lectura fuera de los límites). HeartBleed era un problema de seguridad de la memoria (lectura fuera de los límites). Stagefright también en Android (escrituras fuera de los límites). ¿ La vulnerabilidad Ghost en glibc? Puedes apostar (escritura fuera de los límites).

Estas vulnerabilidades y exploits, y muchos otros, son posibles porque C y C++ no son seguros para la memoria. Las organizaciones que escriben grandes cantidades de C y C++ inevitablemente producen una gran cantidad de vulnerabilidades que pueden atribuirse directamente a la falta de seguridad de la memoria. Estas vulnerabilidades se explotan, poniendo en peligro los hospitales , los disidentes de derechos humanos y los expertos en políticas de salud . Usar C y C++ es malo para la sociedad , malo para su reputación y malo para sus clientes .

## ¿Qué otros problemas están asociados con los idiomas que no son seguros para la memoria?

Los lenguajes que no son seguros para la memoria también tienen un impacto negativo en la estabilidad, la productividad del desarrollador y el rendimiento de la aplicación.

Debido a que los lenguajes que no son seguros para la memoria tienden a permitir más errores y bloqueos, la estabilidad de la aplicación puede verse muy afectada. Incluso cuando los bloqueos no son sensibles a la seguridad, siguen siendo una experiencia muy mala para los usuarios.

Peor aún, estos errores pueden ser increíblemente difíciles de rastrear para los desarrolladores. La corrupción de la memoria a menudo puede causar fallas muy lejos de donde realmente está el error. Cuando se trata de subprocesos múltiples, se pueden desencadenar errores adicionales por ligeras diferencias en qué subproceso se ejecuta cuándo, lo que lleva a errores aún más difíciles de reproducir. El resultado es que los desarrolladores a menudo necesitan mirar los informes de fallas durante horas para determinar la causa de un error de corrupción de memoria. Estos errores pueden permanecer sin corregir durante meses, con los desarrolladores absolutamente convencidos de que existe un error, pero sin tener idea de cómo avanzar para descubrir su causa y solucionarlo.

Por último, está el rendimiento. En décadas pasadas, uno podía confiar en que las CPU se volvían significativamente más rápidas cada uno o dos años. Este ya no es el caso. En cambio, las CPU ahora vienen con más núcleos. Para aprovechar los núcleos adicionales, los desarrolladores tienen la tarea de escribir código de subprocesos múltiples.

Desafortunadamente, los subprocesos múltiples exacerban los problemas asociados con la falta de seguridad de la memoria. Como resultado, los esfuerzos para aprovechar las CPU de varios núcleos a menudo son intratables en C y C++. Por ejemplo, Mozilla tuvo múltiples intentos fallidos de introducir subprocesos múltiples en el subsistema CSS de C++ de Firefox antes de finalmente (con éxito) reescribir el sistema en Rust de subprocesos múltiples.

## ¿Cuál es el camino correcto a seguir?

¡Use lenguajes seguros para la memoria! Hay un montón de grandes para elegir. ¿Escribir un kernel de sistema operativo o un navegador web? ¡Considera el óxido! ¿Construir para iOS y macOS? Swift te tiene cubierto. servidor de red? Ir es una buena elección. Estos son solo

algunos ejemplos, hay muchos otros excelentes lenguajes seguros para la memoria para elegir (¡y muchas otras maravillosas combinaciones de casos de uso!).

Cambiar el lenguaje de programación que usa su organización no es algo que deba tomarse a la ligera. Significa cambiar las habilidades que está buscando cuando contrata, significa volver a capacitar a su fuerza laboral, significa reescribir grandes cantidades de código. No obstante, creemos que esto es necesario a largo plazo, por lo que nos gustaría explicar por qué las alternativas a la adopción de un nuevo lenguaje de programación no han tenido éxito.

Si damos por sentado que el uso de un lenguaje inseguro producirá una serie de vulnerabilidades, la pregunta que nos gustaría hacer es: ¿existen técnicas que podamos emprender para reducir este riesgo, sin obligarnos a cambiar por completo los lenguajes de programación? Y la respuesta es absolutamente sí. No todos los proyectos escritos en lenguajes inseguros son igualmente inseguros y poco confiables.

Algunas prácticas que pueden reducir el riesgo de usar un lenguaje inseguro son:

- Usar algunos modismos modernos de C++ que pueden ayudar a producir un código más seguro y confiable
- Uso de fuzzers y desinfectantes para ayudar a encontrar errores antes de que entren en producción
- Uso de mitigaciones de exploits para ayudar a aumentar la dificultad de explotar vulnerabilidades
- Separación de privilegios para que, incluso cuando se explota una vulnerabilidad, el radio de explosión sea menor

Estas prácticas reducen significativamente el riesgo de usar un lenguaje inseguro, y si no logramos convencerlo de que cambie de lenguaje y va a continuar escribiendo C y C++, adoptarlos es imperativo. Desafortunadamente, también son lamentablemente insuficientes.

Las personas que están a la vanguardia en el desarrollo de modismos, fuzzers, desinfectantes, mitigaciones de exploits y técnicas de separación de privilegios modernos de C++ son desarrolladores de navegadores y sistemas operativos, precisamente los grupos que destacamos al principio con estadísticas sobre la prevalencia de problemas de seguridad de la memoria. A pesar de la inversión de estos equipos en estas técnicas, el uso de lenguajes inseguros los agobia. En pwn2own, una gran competencia de piratería, en 2019 más de la mitad de las vulnerabilidades explotadas en estos productos se debieron a la falta de seguridad de la memoria y, con una excepción, cada ataque exitoso explotó al menos una vulnerabilidad de seguridad de la memoria.

## ¿Es realmente práctico abandonar C y C++?

Con suerte, ya lo hemos convencido de que los lenguajes inseguros como C y C ++ son las causas fundamentales de la gran parte de la inseguridad en nuestros productos, y que si bien hay prácticas que puede emprender para reducir el riesgo, no puede acercarse. a eliminarlo. Todo lo cual puede dejarlo con la sensación de que cambiar el lenguaje de programación que usa para producir millones de líneas de código es una tarea abrumadoramente grande. Al

dividirlo en partes manejables, podemos comenzar a progresar: nuestro objetivo no es reescribir el mundo a lo grande, sino avanzar hacia la reducción de nuestro riesgo.

El primer lugar para comenzar es con nuevos proyectos. Para estos, tiene la opción de simplemente usar un lenguaje seguro para la memoria. Estos tienen el riesgo más bajo, porque no necesita comenzar reescribiendo ningún código, aunque proyectos como este a menudo requieren mejoras en la infraestructura de prueba o implementación para admitir un nuevo lenguaje de programación. Este fue el enfoque adoptado en CrosVM de ChromeOS, un nuevo componente del sistema operativo.

Si no tiene nuevos proyectos, el siguiente lugar para buscar oportunidades para usar un lenguaje seguro para la memoria son los nuevos componentes de un proyecto existente. Varios de los lenguajes seguros para la memoria tienen soporte de primera clase para interoperar con bases de código C y C++ (tanto Rust como Swift, por ejemplo). Esto requiere una inversión inicial ligeramente mayor, ya que requiere la integración en los sistemas de compilación, así como la creación de abstracciones en un nuevo lenguaje para objetos y datos que deben pasar a través de la frontera entre los dos lenguajes. Esta es la estrategia que se utilizó con éxito cuando se implementó WebAuthn como un nuevo componente de Firefox y por un proyecto para habilitar la escritura de módulos del kernel de Linux en Rust .

Lo que estos dos primeros enfoques tienen en común es que tratan con código nuevo. Esto tiene la ventaja de tener puntos de interacción bien definidos con el código existente y no es necesario volver a escribir nada para comenzar el esfuerzo. También le brinda la oportunidad de detener el sangrado: no hay componentes nuevos en lenguajes inseguros, y nos ocuparemos del código existente de forma incremental. Para los proyectos que no tienen ningún componente nuevo natural para empezar a usar un lenguaje seguro para la memoria, la adopción es más desafiante.

En este caso, debe buscar algún componente existente para *reescribir* de un idioma no seguro a un idioma seguro. Es mejor si el componente que elige es algo en lo que ya estaba considerando una reescritura: tal vez por rendimiento, por seguridad o porque el código era demasiado difícil de mantener. Debe intentar elegir algo con el alcance más pequeño posible para su primera reescritura de seguridad de la memoria, para ayudar a que el proyecto tenga éxito y se envíe lo más rápido posible; esto ayuda a minimizar el riesgo inherente a una reescritura. Stylo, la reescritura del motor CSS de Firefox en Rust, es un ejemplo exitoso de este enfoque.

Independientemente de qué enfoque sea el más adecuado para su organización, hay algunas cosas que debe tener en cuenta para maximizar sus posibilidades de éxito. El primero es asegurarse de tener campeones internos e ingenieros senior que puedan proporcionar revisiones de código y tutoría en un lenguaje que será nuevo para muchos miembros del equipo. La extensión natural de esto es asegurarse de que los ingenieros que trabajarán en un nuevo idioma tengan recursos disponibles como libros, capacitaciones o guías internas. Finalmente, querrá asegurarse de tener la misma infraestructura compartida para su nuevo idioma que tiene para el anterior, como sistema de compilación, prueba, implementación, informes de fallas y otras integraciones.

## Conclusión

Adoptar un nuevo lenguaje de programación y comenzar el proceso de migración a él no es tarea fácil. Requiere planificación, recursos y, en última instancia, una inversión de toda su organización. La vida sería mucho más fácil si no tuviéramos que contemplar tales cosas. Desafortunadamente, una revisión de los datos deja en claro que simplemente no podemos considerar continuar usando lenguajes inseguros para proyectos sensibles a la seguridad.

Los datos confirman, una y otra vez, que cuando los proyectos usan lenguajes inseguros como C y C++, se ven afectados por una avalancha de vulnerabilidades de seguridad. No importa cuán talentosos sean los ingenieros, cuán grande sea la inversión en la reducción de privilegios y la mitigación de exploits, el uso de un lenguaje que no es seguro para la memoria simplemente genera demasiados errores. Estos errores reducen en gran medida la seguridad, así como la estabilidad y la productividad.

Afortunadamente, no necesitamos estar satisfechos con el statu quo. Los últimos años han producido una oleada de fantásticas alternativas a C y C++, como Rust, Swift y Go, entre muchas otras. Y esto significa que no tenemos que usar las vulnerabilidades de corrupción de la memoria como un lastre alrededor de nuestros cuellos durante años y años, siempre y cuando decidamos no hacerlo. Esperamos que llegue el momento en que elegir usar un lenguaje inseguro se considere tan negligente como no tener autenticación de múltiples factores o no cifrar los datos en tránsito.

## Gracias a Alex Gaynor

Esta explicación se basa, con permiso, en la publicación de blog de Alex Gaynor Introducción a la inseguridad de la memoria para vicepresidentes de ingeniería .

---

1. Esta es específicamente una medida de vulnerabilidades de software, no incluye cosas como el phishing de credenciales, que son increíblemente comunes. ↩




548 Market St, PMB 57274
San Francisco, California 94104-5401

Envíe todo correo o consultas a:
PO Box 18666 , Minneapolis , MN 55418-0666 , EE . UU.

Política de privacidad *

◯ Acepto que ISRG maneje mi información como se indica en la política de privacidad .

◯ No, no acepto que ISRG maneje mi información como se indica en la política anterior (no estará suscrito a esta lista).

Confirme cómo le gustaría recibir noticias de ISRG:

☐ Correo electrónico

Puede darse de baja en cualquier momento haciendo clic en el enlace en el pie de página de nuestros correos electrónicos. Usamos Mailchimp como nuestra plataforma de marketing. Al hacer clic a continuación para suscribirse, reconoce que su información se transferirá a Mailchimp para su procesamiento. Obtén más información sobre las prácticas de privacidad de Mailchimp aquí.

**Suscribir**


Volver arriba